IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Gerard Chauvel, et al.	§	Confirmation No.:	1111
		§		
		§	Group Art Unit:	2188
Serial No.:	10/632,215	§		
		§	Examiner:	H.S. Ahmed
Filed:	July 31, 2003	§		
		§	Atty. Docket No.:	TI-35460
For:	Reformat Logic To Translate Between A Virtual Address And A Compressed Physical Address	§		(1962-05417)
		§		
		§		
		§		
		§		

APPEAL BRIEF

Mail Stop Appeal Brief—Patents
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Date: June 30, 2008

Sir:

Appellants hereby submit this Appeal Brief in connection with the above-identified application. A Notice of Appeal was electronically filed on May 13, 2008.

TABLE OF CONTENTS

I. REAL PARTY IN INTEREST

The real party in interest is Texas Instruments Incorporated, a Delaware corporation, having its principal place of business in Dallas, Texas.

II. RELATED APPEALS AND INTERFERENCES

Appellants are unaware of any related appeals or interferences.

III. STATUS OF THE CLAIMS

Originally filed claims:	1-26
Claim cancellations:	None
Added claims:	None
Presently pending claims:	1-26
Presently appealed claims:	1-26

IV. STATUS OF THE AMENDMENTS

No claims were amended after the Final Office action dated February 15, 2008.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Many types of devices require device drivers for the operation of the device. Such devices may include displays and keyboards. A device driver typically requires a portion of memory to be allocated for its use in providing data to or receiving data from the device it controls. With regard to at least some high level languages (e.g., Java), such languages typically require a "call" to a device driver that may be written in a "native" language such as C. The high level application may use a data structure to provide data to, or receive data from, a corresponding data structure in the device driver memory. The two data structures may not be directly compatible and thus, a mapping between the two may be needed. Mapping a data structure from a high level language to the data structure in the device driver memory can be computationally intensive. Additionally, the calls that permit the context change between the high level application and the device driver undesirably introduce latency. See specification page 3 line 1 of para. [0003] through line 8 of para. [0004]. Appellants have addressed this problem.

In accordance with the invention of claim 1, a system[1] comprises a processor[2] executing an application, a peripheral device[3] coupled to the processor, and memory[4] containing an application data structure[5] accessible by the application. Accesses to the application data structure and accesses to the device are formatted differently.[6] Data can be written to, or read from, the peripheral device via the application data structure.[7] The system also comprises reformat logic[8] coupled to the processor and memory. The reformat logic dynamically reformats an access from the application targeting the

[1] E.g. Fig. 1, system 100. Page 6 line 1 of para. [0019].

[2] E.g. Fig. 1, JSM 102. Page 6 line 3 of para. [0019].

[3] E.g. Fig. 1, display 114. Page 6 line 2 of para. [0020].

[4] E.g. Fig. 1, memory 106. Page 6 line 5 of para. [0019].

[5] E.g. Fig. 4, array 152.

[6] Page 8 lines 1-9 of para. [0024].

[7] Page 8 lines 5 of para. [0024].

[8] E.g. Fig. 2, reformat logic 154. Page 7 line 7 of para. [0022].

application data structure to a format that comports with the device,[9] thereby permitting the application to manage the peripheral device without the use of a device driver. The reformat logic includes alignment logic that implements a read-modify-write operation to write a value from the application data structure across byte boundaries in the display buffer.[10]

In accordance with the invention of claim 15, reformat logic[11] comprises a plurality of registers[12] and translation logic[13] that accesses the registers and that receives a memory access targeting an application data structure[14] that has a different format[15] than for accesses that are provided to a device[16] that is external to the reformat logic and that reformats the request to a format compatible with the device based on values stored in the registers. The translation logic implements a read-modify-write operation to write a value from the application data structure across byte boundaries of a memory buffer associated with the device.[17]

In accordance with the invention of claim 22, a method comprises receiving a physical address from a processor,[18] the physical address associated with an application data structure.[19] The method further comprises converting the physical address to a device buffer address associated with a device buffer, the device buffer being accessed

---

[9] Page 9 lines 4-7.

[10] Page 9 lines 7-10.

[11] E.g. Fig. 2, reformat logic 154. Page 7 line 7 of para. [0022].

[12] E.g. Fig. 4, configuration registers 157. Page 11 lines 3-4 of para. [0030].

[13] E.g. Fig. 4, translation logic 159. Page 11 line 4 of para. [0030].

[14] E.g. Fig. 4, array 152. Page 7 lines 2-3 of para. [0022].

[15] Page 8 lines 1-9 of para. [0024].

[16] E.g. Fig. 1, display 114. Page 6 line 2 of para. [0020].

[17] Page 9 lines 7-10.

[18] E.g. Fig. 1, JSM 102. Page 6 line 3 of para. [0019].

[19] E.g. Fig. 4, array 152. Page 7 lines 2-3 of para. [0022].

with a different number of bits than the application data structure.[20] The method also comprises providing the converted device buffer address to the device buffer to permit the processor to control a peripheral device[21] without using a driver associated with the peripheral device. The method further comprises performing a read-modify-write operation to write a value from the application data structure to the device buffer across byte boundaries of the device buffer.[22]

[20] Page 8 lines 1-9 of para. [0024].

[21] E.g. Fig. 1, display 114. Page 6 line 2 of para. [0020].

[22] Page 9 lines 7-10.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Whether claims 1-10 and 12-26 are anticipated under 35 U.S.C. 102(b) by Lehman (U.S. Pat. No. 5,680,161).

Whether claim 11 is obvious under 35 U.S.C. 103(a) over Lehman in view of Huber (U.S. Pat. No. 6,070,173).

VII. ARGUMENT

A. Overview of Lehman

In Figure 2, Lehman teaches loading 32-bit video memory with 24-bits of pixel color values. Because each 24-bit color value naturally does not require 32 bits, each 32-bit row stores portions of color values of at least two different pixels. For example, the first row (MA 0) is shown stored with the red, green and blue values for pixel 0 (R0, G0, B0) along with the red value for the next pixel (R1). The remainder of pixel 1's color values (G1 and B1) are stored beginning in the next row (MA 1). Lehman teaches sequentially loading the video memory in this manner. See col. 5, lines 29-58. Once the video memory is loaded, the data can be read from the memory in an appropriate way to recreate the RGB values for each pixel. Lehman loads the color values sequentially and does not teach writing only to a portion of an already-written memory address. That is, Lehman does not teach updating just one 8-bit color value at a given 32-bit memory address. Because Lehman has no such need of partial memory address updating, Lehman has no need for, and does not teach, performing a read-modify-write operation.

B. Claims 1-10 and 12-26

Claim 1 requires that the reformat logic "includes alignment logic that implements a read-modify-write operation to write a value from the application data structure across byte boundaries in the display buffer." Lehman teaches sequentially loading the 32-bit video memory with 24-bit pixel values, appending the next pixel value to the end of the preceding pixel value. Lehman does not teach writing only to a portion of an already-written memory address. Further, Lehman has no need for partial memory address updating. Accordingly, Lehman has no need for, and does not teach, performing a read-modify-write operation. For at least this reasons, the Examiner erred in rejecting claim 1 and its dependent claims over Lehman.

Dependent claim 9 is allowable for an additional reason. Claim 9 requires a plurality of programmable registers that stores values indicative of "starting and ending addresses of the application data structure in which accesses are to be reformatted by the reformat logic, the starting address of the device buffer, n, and m." The value "n"

specifies the bit width of the application data structure and "m" specifies the bit width of the display buffer. The Examiner referred to col. 3, line 55 through col. 4, line 3 for these limitations. That passage in Lehman refers to a "video shift register circuit" and an "output register." The video shift register circuit comprises a pixel buffer into which the pixel color values are stored. The output register also stores pixel color values. Neither of these registers is described as storing what is required by claim 9, that is, "starting and ending addresses of the application data structure in which accesses are to be reformatted by the reformat logic, the starting address of the device buffer, n, and m." Applicants find no teaching elsewhere in Lehman of registers that store the claimed values.

Claim 12 requires that the multiplexer selectively permits accesses from the application to "bypass the reformat logic so that such accesses are not reformatted by the reformat logic." Lehman has no such bypass mechanism. The Examiner referred to col. 12, lines 57-60 of Lehman with regard to claim 12. That passage describes pixel selector 84 which simply selects one of the groups of pixel color values. While pixel selector 84 may be a multiplexer, it is not described as being used to "bypass the reformat logic so that such accesses are not reformatted by the reformat logic." For this additional reason, claim 12 as well as claims 13 and 14 which depend on claim 12 are allowable over the art of record.

Independent claims 15 and 22 also require a "read-modify-write operation" as did claim 1. Thus, those claims and their dependent claims are allowable for much the same reason as articulated above with regard to claim 1.

C. Claim 11

Claim 11 depends from 1 and thus inherits the limitations of claim 1. Claim 1 is patentable over Lehman as explained above. Huber does not satisfy the deficiency of Lehman. As such, the Examiner erred in rejecting claim 11 for much the same reason as for claim 1.

D. Conclusion

For the reasons stated above, Appellants respectfully submit that the Examiner erred in rejecting all pending claims. It is believed that no extensions of time or fees are

required, beyond those that may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required (including fees for net addition of claims) are hereby authorized to be charged to Texas Instruments Incorporated's Deposit Account No. 20-0668.

Respectfully submitted,

/Jonathan M. Harris/

Jonathan M. Harris
PTO Reg. No. 44,144
CONLEY ROSE, P.C.
(713) 238-8000 (Phone)
(713) 238-8008 (Fax)
ATTORNEY FOR APPELLANTS

VIII. CLAIMS APPENDIX

1. (Previously Presented) A system, comprising:

a processor executing an application;

a peripheral device coupled to the processor;

memory containing an application data structure accessible by said application, wherein accesses to said application data structure and accesses to said device are formatted differently, and wherein data can be written to, or read from, the peripheral device via the application data structure; and

reformat logic coupled to the processor and memory, the reformat logic dynamically reformats an access from the application targeting the application data structure to a format that comports with the device, thereby permitting said application to manage the peripheral device without the use of a device driver, wherein the reformat logic includes alignment logic that implements a read-modify-write operation to write a value from the application data structure across byte boundaries in the display buffer.

2. (Original) The system of claim 1 wherein the peripheral device comprises a display.

3. (Original) The system of claim 1 wherein the application data structure comprises an array.

4. (Original) The system of claim 3 wherein the array comprises a multi-dimensional array.

5. (Original) The system of claim 3 wherein array comprises a single-dimensional array.

6. (Original) The system of claim 1 further including a device buffer associated with the device and wherein the application data structure comprises an n-bit data structure and the device buffer comprises an m-bit display buffer, wherein n is different than m, and the reformat logic reformats an n-bit access from the application to an m-bit access for the device buffer.

7. (Previously Presented) The system of claim 6 wherein n is not an integer multiple of m.

8. (Original) The system of claim 6 wherein m is less than n.

9. (Original) The system of claim 6 wherein the reformat logic comprises a plurality of registers which are programmable to store a plurality of values, said values comprising information that is indicative of the starting and ending addresses of the application data structure in which accesses are to be reformatted by the reformat logic, the starting address of the device buffer, n, and m.

10. (Original)The system of claim 6 wherein the reformat logic comprises a plurality of registers which are programmable to store a plurality of values, said values comprising information that is indicative of the starting and ending addresses of the application data structure in which accesses are to be reformatted by the reformat logic, the starting address of the device buffer, n, and value indicative of the ratio between n and m.

11. (Original)The system of claim 6 wherein the registers are programmed by a virtual machine.

12. (Previously Presented) The system of claim 1 further comprising a multiplexer that selectively permits accesses from the application to be provided to bypass the reformat logic so that such accesses are not reformatted by the reformat logic and permits accesses from the application to be reformatted before being provided to the memory.

13. (Original) The system of claim 12 wherein the reformat logic controls the multiplexer to select whether or not a reformatted access is to be provided to the memory.

14. (Original) The system of claim 12 wherein the processor supplies an address to the multiplexer and to the reformat logic and asserts a signal to the multiplexer to cause the multiplexer to select whether or not a reformatted access is to be provided to the memory.

15. (Previously Presented) Reformat logic, comprising:

a plurality of registers; and

translation logic that accesses the registers and that receives a memory access targeting an application data structure that has a different format than for accesses that are provided to a device that is external to said reformat logic and that reformats the request to a format compatible with the device based on values stored in the registers, wherein the translation logic implements a read-modify-write operation to write a value from the application data structure across byte boundaries of a memory buffer associated with said device.

16. (Previously Presented) The reformat logic of claim 15 wherein the application data structure is n-bit accessible and the memory buffer is m-bit accessible where m is less than n, and the translation logic reformats the request from an n-bit format to an m-bit format.

17. (Previously Presented) The reformat logic of claim 16 wherein n is not an integer multiple of m and the reformat logic includes alignment logic to implement the read-modify-write operation.

18. (Original) The reformat logic of claim 16 further comprising a plurality of registers which are configured to be programmed to store a plurality of values, said values comprising values that enable to determine the starting and ending addresses of the application data structure in which accesses are to be reformatted by the reformat logic, the starting address of the memory buffer, n, and m.

19. (Original) The reformat logic of claim 16 further comprising a plurality of registers which are configured to be programmed to store a plurality of values, said values comprising values that enable to determine the starting and ending addresses of the application data structure in which accesses are to be reformatted by the reformat logic, the starting address of the memory buffer, n, and value indicative of the ratio between n and m.

20. (Original) The reformat logic of claim 15 wherein the translation logic reformats both read and write requests.

21. (Original) The reformat logic of claim 15 wherein the memory buffer for which the reformat logic reformats a request comprises a display memory buffer associated with a display.

22. (Previously Presented) A method, comprising:

receiving a physical address from a processor, the physical address associated with an application data structure;

converting the physical address to a device buffer address associated with a device buffer, the device buffer being accessed with a different number of bits than the application data structure;

providing the converted device buffer address to the device buffer to permit the processor to control a peripheral device without using a driver associated with the peripheral device; and

performing a read-modify-write operation to write a value from the application data structure to the device buffer across byte boundaries of the device buffer.

23. (Original) The method of claim 22 wherein receiving the physical address from the processor is performed upon writing to the application data structure.

24. (Original) The method of claim 22 wherein receiving the physical address from the processor is performed upon reading from the application data structure.

25. (Original) The method of claim 22 further including completing a read or write transaction to the device buffer using the converted device buffer address.

26. (Original) The method of claim 22 wherein the device buffer is accessed with fewer bits than the application data structure.

IX. EVIDENCE APPENDIX

None.

X. RELATED PROCEEDINGS APPENDIX

None.